Question 1: Algorithm Analysis and Big O (10 Points)

Give a tight bound of the worst-case runtime complexity class for each of the following code fragments in Big-Oh notation, in terms of the variable *N*. (Write the growth rate as *N* grows.) Write a simple expression that gives only a power of *N*, such as O(N^2) or O(log *N*), not an exact calculation like O($2N^3 + 4N + 14$).
For each problem below, assume that you already have the following five Stanford Library data structures, each with *N* elements, which were already declared as follows (and filled). Each sub-problem has a complete set of elements in those data structures (i.e., the problems are independent, and modifying a container in one sub-problem does not affect the next sub-problem). If there are multiple data structures in one problem, they all have the same *N* number of elements already populated:

- `Map<int, int> myMap;`
- `Set<int> mySet;`
- `Vector<int> myVector;`
- `Stack<int> myStack;`
- `Queue<int> myQueue;`

You also have a **`Grid`** with *N*x*N* elements, defined as follows. *Note:* a Stanford Grid has O(1) read time for all elements, just like a Vector.

- `Grid<int> myGrid(N, N)`

```
while (!myStack.isEmpty()) {
    int x = myQueue.dequeue();
    myQueue.enqueue(x * myStack.pop());
}
```

Answer:

```
for (int i = 0; i < N; i++) {
    int x = myVector.remove(0);
    myVector.add(N - x);
}
```

Answer:

```
int count = 0;
for (int i = 0; i < N; i++) {
    if (mySet.contains(i)) {
        count++;
    }
}
```

Answer:

```
int i = N;
while (i > 1) {
    myQueue.enqueue(i);
    i = i / 2;
}
```

Answer:

```
for (int r = 0; r < N; r++) {
    for (int c = 0; c < N; c++) {
        myMap[myGrid[r][c]] = myGrid[c][r];
    }
}
```

Answer:

```
int value = 1;
for (int i = 1; i < 20 * N; i++) {
    value *= i;
    myVector.add(value);
}

for (int j = 1; j < 200 * N; j++) {
    value *= j;
    myVector.add(value);
}
```

Answer:

Question 2: Wordle

Wordle is a word-guessing game made by Josh Wardle that gained popularity in the past year. Here's how the game works.

There is one word that the user is attempting to uncover, let's call this **`actualWord`**. When a user guesses what **`actualWord`** is, each character in this **`guessedWord`** is put into one of three categories based on its presence in **`actualWord`**:

- Category 0: the character is not in **`actualWord`**
- Category 1: the character is in **`actualWord`** but not in the right position within **`actualWord`**
- Category 2: the character is in **`actualWord`** and is in the right position within **`actualWord`**

Your task is to write a function that scores a user's Wordle guess:

```
int scoreWordleGuess(string guessedWord,
string actualWord);
```

We'll say characters in Category 0 have a score of 0, characters in category 1 have a score of 1, and characters in Category 2 have a score of 2.

For example, **`scoreWordleGuess("smart", "sweat")`** would return 5 since

- **`'s'`** is in Category 2, it's in **`"sweat"`** and in the right position
- **`'m'`** is in Category 0, it's not in **`"sweat"`**
- **`'a'`** is in Category 1, it's in **`"sweat"`** but not at the right position (index 2 within **`"smart"`** and index 3 within **`"sweat"`**)
- **`'r'`** is in Category 0, it's not in **`"sweat"`**
- **`'t'`** is in Category 2, it's in **`"sweat"`** and in the right position

Notes on this problem:

- You can assume that `actualWord` and `guessedWord` are the same length. Both strings are non-empty, and of arbitrary size (not always 5 letters like the actual Wordle game).
- You can assume that there are no duplicate characters in either `actualWord` or `guessedWord`. For example, neither `actualWord` nor `guessedWord` will be `"elder"` since this word has a double `'e'`.
- Your solution should be case-insensitive. This means that `scoreWordleGuess("SMart", "sweAt")` should also return 5.

Answer:

Report the Big-O runtime of your solution:

There is a solution to scoreWorldeGuess that runs in **`O(n log n)`** time. If your solution does that, explain in 50 words why that's the case. If not, please explain how you would modify your program to do so (you do not have to modify your code, a working solution will get full credit to the first part of this question).

Answer:

Question 3: Non-recursive “balanced operators" with a Stack

For assignment 3, you had to write a recursive function to determine whether a bracketed string was nested and matched ("balanced"). Now we're going to have you implement an iterative version using a stack.
Here is the basic algorithm (without the low-level details you will need to figure out):

1. Iterate through the string, checking each character.
2. If you find an opening bracket, push it onto the stack.
3. If you find a closing bracket, pop the last value off the stack. If the stack was empty, or if the current character and the character popped off the stack are not matching brackets, the string is not balanced.
4. If you reach the end of the string and there are still values on the stack, the string is not balanced.

Given a **`Map<char, char>`** of matching brackets, with the keys as the left brackets, and the values as the corresponding right brackets (see the example code below), and a **`string`**, write the function **`matchedOperators`**, as shown below. Your code should run in **`O(n)`** time, where **`n`** is the length of the string.

Note:: The **`Map`** can be *any* mapping of left-bracket/right-bracket. In other words, what constitutes a "bracket" is defined by the map. Your solution should not assume that the brackets are the same as in the homework problem. You can assume that there are unique pairs of brackets (e.g., all values in the map will be unique). In the example below, curly braces, square brackets, the colon / questionmark pair, and the **`'|'`** and **`'!'`** pair are brackets, but parentheses are not considered brackets.

```
// returns true if the string is balanced, false if it is not balanced
bool matchedOperators(Map<char, char> operators, string s);
```

```
// Example on how to use:
int main() {
    Map<char, char> ops;
    ops['{'] = '}'; // opening bracket {, closing bracket }
    ops['['] = ']'; // opening bracket [, closing bracket ]
    ops[':'] = '?'; // opening bracket :, closing bracket ?
    ops['|'] = '!'; // opening bracket |, closing bracket !
    cout << std::boolalpha;
    // prints true
    cout << matchedOperators(ops, ":4 + 5? * {6 + [4-3]}") << endl;

    // prints false
    cout << matchedOperators(ops, "|4 + 5!! * {6 + [4-3]}") << endl;

    // prints false
    cout << matchedOperators(ops, "{ [ x } y ]") << endl;

    // prints true
    cout << matchedOperators(ops,
          " int main() { int x = 2 * (vec[2] + 3); x = (1 + random()); }")
          << endl;
}
```

Answer:

Question 4: The Logistic Map

The *logistic map* is a non-linear equation that demonstrates how chaotic behavior can arise in a simple way. It has a recursive defintion as follows:

$$x_{n+1} = rx_n\left(1 - x_n\right)$$

This creates a sequence of values given by x_0, x_1, x_2, ..., x_{n-1}, x_n. Both r and x_0 are constants, with x_0 defined as the initial condition.

Write a recursive function to calculate x_n when given the values r and x_0:

```
double logistic(int n, double r, double x0);
```

Notes:

- While you can solve this with two recursive calls, your solution should only have one for full credit to avoid an unnecessary call.
- **`x0`** is the return value of the function when **`n`** equals zero.
- Neither **`r`** nor **`x0`** change while the function is running.

Answer:

Question 5: More recursion

Part A: Given a string, write a recursive function, **`dashSep`** to return a string where all adjacent characters are separated by a dash.
Examples:

- **`dashSep("hello") -> "h-e-l-l-o"`**
- **`dashSep("dogs") -> "d-o-g-s"`**
- **`dashSep("abc") -> "a-b-c"`**
- **`dashSep("ab") -> "a-b"`**
- **`dashSep("a") -> "a"`**
- **`dashSep("") -> ""`**

```
string dashSep(string s);
```

Answer:

Part B: Given a string, write a recursive function, **`twinsDash`** to return a string where all characters that are the same are separated by a dash.
Examples:

- **`twinsDash("hello") -> "hel-lo"`**
- **`twinsDash("xxyy") -> "x-xy-y"`**
- **`twinsDash("aaaa") -> "a-a-a-a"`**

```
string twinsDash(string s);
```

Answer: